AMENDMENTS TO THE SPECIFICATION

On page 1, paragraph 2:

[001] Media applications have been driving microprocessor development for more than a decade. In fact, most computing upgrades in recent years have been driven by media applications, predominantly within the computer segments, but also in enterprise segments for entertainment-enhanced education and communication purposes. Nevertheless, future media applications will require even higher computational requirements. As a result, tomorrow's personal computer (PC) will be even richer in audio-visual effects, as well as being easier to use and more importantly, computing will ~~merger~~ merge with communications.

On page 5, paragraph 32:

[0032] In one embodiment, addresses of the data elements are contained within a data storage device and indicated as index addresses. In addition, the data elements are stored ~~n~~ in one or more data storage areas of a memory device, which include look-up tables, data arrays or the like. In addition, data elements within the destination data storage device, as well as address indexes within the address data storage device may be organized in response to execution of a data shuffle instruction according to a data processing operation instruction.

On page 6, paragraph 40:

[0040] Execution unit 230 operates on packed data, according to the instructions received by processor ~~109~~ 209 that are included in multiple-data instruction set 240. Execution unit 230 also operates on scalar data according to instructions implemented in general-purpose processors. Processor 209 is capable of supporting the Pentium® microprocessor instruction set and the multiple-data instruction set 240. By including packed instruction set 240 in a standard microprocessor instruction set, such as the Pentium® microprocessor instruction set, packed data instructions can be easily incorporated into existing software (previously written for the standard

microprocessor instruction set). Other standard instruction sets, such as the PowerPC™ and the Alpha™ processor instruction sets may also be used in accordance with the described invention. (Pentium® is a registered trademark of Intel Corporation. PowerPC™ is a trademark of IBM, APPLE COMPUTER and MOTOROLA. Alpha™ is a trademark of Digital Equipment Corporation.)

On page 7, paragraph 44:

[0044] Still referring to FIG. 2, the computer system 200 of the present invention may include a display device 221 such as a monitor. The display device 221 may include an intermediate device such as a frame buffer. The computer system 200 also includes an input device 222 such as a keyboard, and a cursor control 223 such as a mouse, or trackball, or trackpad. The display device 221, the input device 222, and the cursor control 223 are coupled to bus 201. Computer system ~~100~~ 200 may also include a network connector 224 such that computer system 200 is part of a local area network (LAN) or a wide area network (WAN).

On page 8, paragraph 46:

[0046] FIG. 3 illustrates a detailed diagram of processor ~~309~~209. Processor ~~309~~ 209 can be implemented on one or more substrates using any of a number of process technologies, such as, BiCMOS, CMOS, and NMOS. Processor 209 comprises a decoder 265 for decoding control signals and data used by processor 209. Data can then be stored in register file 300 via internal bus 270. As a matter of clarity, the registers of an embodiment should not be limited in meaning to a particular type of circuit. Rather, a register of an embodiment need only be capable of storing and providing data, and performing the functions described herein.

On page 9, paragraph 50:

[0050] ~~Functional~~ Execution unit 230 may include one or more function units to performs the operations carried out by processor 209. Such operations may include shifts, addition, subtraction and multiplication, multiply-accumulate, etc. ~~Functional~~ Execution unit 230

connects to internal bus 270. Cache 260 is an optional element of processor 209 and can be used to cache data and/or control signals from, for example, main memory ~~104~~204. Cache 260 is connected to decoder 265, and is connected to receive control signal 207. However, in contrast to conventional processors, one embodiment of the processor includes a dedicated memory device 320. The dedicated memory device contains computationally intensive operation results that are calculated before an application start-up or at application initialization. Representatively, data shuffle unit 232 may be coupled between register file 300 and execution unit 230. In one embodiment, data shuffle unit 232 organizes, in response to execution of an address shuffle instruction, address elements within the address data storage device 310 according to a data processing operation; and organizes, in response to executing a data shuffle instruction, data elements within destination data storage device 310 according to a data processing operation.

On page 10, paragraph 56:

[0056] In contrast to conventional data access methods, which include a capability to load one or multiple bytes of data that are stored in a contiguous memory space, the processor 209 as depicted in FIGS. 6A includes multiple-data ~~operation~~ instruction set 240. The multiple-data operation set enables loading of data within destination register R_1 310-2, that is initially randomly distributed within storage areas 330 (330-1, . . ., 330-N) of the dedicated memory device 320. Accordingly, once computationally intensive operations are stored within storage areas 330, an application utilizing or requiring the computational results ~~will~~ loads the desired results within a destination register 310-2 based on, for example, address indexes which are stored in address register (R_0) 310-1.

[0056.1] ~~As such, the~~In one embodiment, execution unit 230 ~~will~~ receives an address index, as well as other information~~, in order~~ to access data elements stored within the storage areas 330 and stores those requested data elements within the destination register 310-1. Representatively, data shuffle unit 232 is coupled between register file 300 and the execution unit. In one embodiment, data shuffle unit 232 organizes, in response to execution of an address shuffle instruction, address elements within the address data storage device 310 according to a data

processing operation and organizes, in response to executing a data shuffle instruction, data elements within destination data storage device 310-2 according to a data processing operation.

On page 10, paragraph 57:

[0057] In one embodiment, a multiple move operation: MOV_MULTIPLE R_0, [R_1, offset, tableN] directs the loading of multiple, randomly distributed data ~~within~~ from the dedicated memory device 320 as depicted in FIGS. 6B and 6C. Accordingly, the R_0 argument indicates the address register 310-1 which contains address indexes for capturing data within the storage areas 330, which in one embodiment are configured as look-up tables. The R_1 argument indicates the destination register 310-2 wherein to store the data elements which are read from the storage areas 330 of the memory device 320. The offset argument indicates an area 334 (334-1, . . ., 334-N) within which the look-up table 330 has been divided. Finally, the tableN argument 336 indicates the table 330 within the memory device 320 which contains the data. For example, FIG. 6C depicts a 1024 byte table 330 with a 256-byte address index 332 and a two bit offset 334.

On page 11, paragraph 58:

[0058] Loading of multiple randomly distributed data is further illustrated with reference to FIG. 7. Referring to FIG. 7, FIG. 7 depicts the R_0 address register 310–1, which is indexed with byte address indexes. Accordingly, the execution unit 230 will select an address index from the R_0 register 310-1 and store a data element read from the address index within the R_1 destination register 310-2. In one embodiment, both the R_0 address register 310 and the destination register 310-2 are, for example, Intel® MMX one hundred twenty-eight bit registers. Accordingly, in response to execution of multiple data load instruction, the present invention teaches the ability to load multiple randomly distributed data ~~within~~ from look-up table 320 ~~within~~ to the destination data storage device 310.

On page 12, paragraph 66:

[0066] Referring now to FIG. 9, FIG. 9 depicts a block diagram illustrating an embodiment wherein the address index data type is a 16-bit word, while the memory access data type is a byte. As such, loading of the multiple randomly distributed data does not completely fill the R_1 destination register 310-2, even following loading of each address index from the R_0 address register 310-1. As recognized by those skilled in the art, the embodiment described with reference to FIG. 9 occurs in forward error control algorithms which utilize Reed Solomon codes. In each case, the position of the index gives the position of the ~~access~~ accessed data ~~in~~ within the destination register.

On page 12, paragraph 67:

[0067] In the embodiments described, indices used for accessing data are stored into the base of the register. In an alternate embodiment, indices are identified by using an operand in the instruction that specifies the offset from the beginning of the index register of the first index that is used for accessing data. However, as depicted in FIG. 9, an alternate solution for selecting the position for loading data in the destination register uses an operand in the instruction to specify the offset from the base of the destination register. Accordingly, as depicted in FIG. 8, data elements with a 32-bit word data type that are indexed with byte address indexes may utilize multiple destination registers in order to store the entire data. In addition, in the embodiment depicted in FIG. 9, smaller index address registers may be utilized in order to load byte data that fills the entire destination data storage device.

On page 14, paragraph 73:

[0073] Finally, referring to FIG. 11, FIG. 11 depicts a data shuffle instruction for organizing addresses within an address storage device and data within a destination data storage device. In the embodiment described, data is organized according to a mask 350. Representatively, data shuffle unit 232 (FIGS. 3 and 6A) organizes, in response to execution of

an address shuffle instruction, address elements within the address data storage device 310 according to mask 350 or organizes, in response to executing a data shuffle instruction, data elements within the destination data storage device 310 according to mask 350.

On page 14, paragraph 74:

[0074] ~~Accordingly, utilizing the assembly language mnemonics provided by the present invention, sample code for Rijndael implementations and for forward error control implementations is provided in Appendices A and B for a Rijndael implementation.~~ The present invention, utilizing the multiple load instruction set as described above, utilizes one conventional load and four non-contiguous loads, which is compared to forty conventional loads in existing methodologies in computing architectures for performing Rijndael implementations. For forward error control implementations, the present invention, using the multiple load instruction set, utilizes two conventional loads, one non-contiguous load and one conventional store instruction, whereas a conventional implementation of forward error control will utilize 48 conventional loads and 16 conventional stores within the conventional computing architectures. Procedural methods for implementing the teachings of the present invention are now described.

On page 14, paragraph 75:

[0075] Referring now to FIG. 12, FIG. 12 depicts a flowchart illustrating a method 400 for performing a multiple data load instruction, for example within the computer system as depicted in FIGS. 2, 3 and 6A-6C in accordance with an embodiment of the present invention. At process block 420, it is determined whether a multiple data load instruction has been executed. In response to execution of a multiple data load instruction, at process block 422, randomly located data from a memory device is loaded within a destination data storage device. Next, a process block 490, it is determined whether a data processing operation has been executed. Finally, at process block ~~520~~500, in response to execution of a data processing operation, data elements within the destination data storage device are processed according to the data processing operation. Once completed, the method terminates.

On page 15, paragraph 80:

[0080] Referring now to FIG. 15, FIG. 15 depicts a flowchart illustrating additional method 432 for reading data elements of process block 430, as depicted in FIG. 14. At process block 432, it is determined whether a data location value of a look-up table containing the data elements has been received. Once received, at process block 436, it is determined whether an offset value indicating a data region within the look-up table is received. Once received, an address index is selected from the address storage device, at process block 438. Once selected, at process block 440, a data element is read from the data region of the look-up table according to the selected address index. Next, at process block 442, process blocks 438 and 440 are repeated for each address index within the address data storage device. Once completed, control flow returns to process block 430, as depicted in FIG. 14.

On page 16, paragraph 81:

[0081] Referring now to FIG. 16, FIG. 16 depicts a flowchart illustrating an additional method 450 for reading data elements for each address within the address storage device of process block 430 within a conventional cached memory or other memory device. At process block 452, it is determined whether a base address value of a data storage area within a memory device containing the data elements is received. Once received, at process block 454, it is determined whether an offset value of a data region within the memory device containing the data elements is received. Once received, an address index from the address storage device is selected, at process block 456.